Modeling and Analysis of Hybrid Systems

What’s decidable about hybrid automata?

Prof. Dr. Erika Ábrahám

Informatik 2 - Theory of Hybrid Systems
RWTH Aachen University

SS 2015
Henzinger et al.: What’s decidable about hybrid automata?
Motivation

- The special class of timed automata with TCTL is decidable, thus model checking is possible.
- What about more expressive model classes for hybrid systems?
Two central problems for the analysis of hybrid automata:

- **Safety**: The problem to decide whether something “bad” can happen during the execution of a system.

- **Liveness**: The problem to decide whether there is always the possibility that something “good” will eventually happen during the execution of a system.

Both problems are decidable in certain special cases, and undecidable in certain general cases.
A particularly interesting class:
What is decidable about hybrid automata?

A particularly interesting class:

- all conditions, effects, and flows are described by rectangular sets.
What is decidable about hybrid automata?

A particularly interesting class:
- all conditions, effects, and flows are described by rectangular sets.

Definition

- A set $\mathcal{R} \subset \mathbb{R}^n$ is rectangular if it is a cartesian product of (possibly unbounded) intervals, all of whose finite endpoints are rationals.
- The set of rectangular sets in $\mathbb{R}^n$ is denoted $\mathcal{R}^n$. 
A rectangular automaton \( \mathcal{A} \) is a tuple
\( \mathcal{A} = (\text{Loc}, \text{Var}, \text{Con}, \text{Lab}, \text{Edge}, \text{Act}, \text{Inv}, \text{Init}) \) with

- finite set of locations \( \text{Loc} \),
- finite set of real-valued variables \( \text{Var} = \{x_1, \ldots, x_n\} \),
- a function \( \text{Con} : \text{Loc} \to 2^{\text{Var}} \) assigning controlled variables to locations,
- finite set of synchronization labels \( \text{Lab} \),
- finite set of edges \( \text{Edge} \subseteq \text{Loc} \times \text{Lab} \times \mathbb{R}^n \times \mathbb{R}^n \times 2^{\{1, \ldots, n\}} \times \text{Loc} \),
- a flow function \( \text{Act} : \text{Loc} \to \mathbb{R}^n \),
- an invariant function \( \text{Inv} : \text{Loc} \to \mathbb{R}^n \),
- initial states \( \text{Init} : \text{Loc} \to \mathbb{R}^n \).
Rectangular automaton

Definition

A rectangular automaton $A$ is a tuple $H = (\text{Loc}, \text{Var}, \text{Con}, \text{Lab}, \text{Edge}, \text{Act}, \text{Inv}, \text{Init})$ with

- finite set of locations $\text{Loc}$,
- finite set of real-valued variables $\text{Var} = \{x_1, \ldots, x_n\}$,
- a function $\text{Con} : \text{Loc} \to 2^{\text{Var}}$ assigning controlled variables to locations,
- finite set of synchronization labels $\text{Lab}$,
- finite set of edges $\text{Edge} \subseteq \text{Loc} \times \text{Lab} \times \mathbb{R}^n \times \mathbb{R}^n \times 2^{\{1, \ldots, n\}} \times \text{Loc}$,
- a flow function $\text{Act} : \text{Loc} \to \mathbb{R}^n$,
- an invariant function $\text{Inv} : \text{Loc} \to \mathbb{R}^n$,
- initial states $\text{Init} : \text{Loc} \to \mathbb{R}^n$.

- **States:** $\sigma = (l, \bar{x}) \in (\text{Loc} \times \mathbb{R}^n)$ with $\bar{x} \in \text{Inv}(l)$
Definition

A rectangular automaton $A$ is a tuple $\mathcal{H} = (\text{Loc}, \text{Var}, \text{Con}, \text{Lab}, \text{Edge}, \text{Act}, \text{Inv}, \text{Init})$ with

- finite set of locations $\text{Loc}$,
- finite set of real-valued variables $\text{Var} = \{x_1, \ldots, x_n\}$,
- a function $\text{Con} : \text{Loc} \rightarrow 2^{\text{Var}}$ assigning controlled variables to locations,
- finite set of synchronization labels $\text{Lab}$,
- finite set of edges $\text{Edge} \subseteq \text{Loc} \times \text{Lab} \times \mathbb{R}^n \times \mathbb{R}^n \times 2^{\{1, \ldots, n\}} \times \text{Loc}$,
- a flow function $\text{Act} : \text{Loc} \rightarrow \mathbb{R}^n$,
- an invariant function $\text{Inv} : \text{Loc} \rightarrow \mathbb{R}^n$,
- initial states $\text{Init} : \text{Loc} \rightarrow \mathbb{R}^n$.

- **States:** $\sigma = (l, \vec{x}) \in (\text{Loc} \times \mathbb{R}^n)$ with $\vec{x} \in \text{Inv}(l)$
- **State space:** $\Sigma \subseteq \text{Loc} \times \mathbb{R}^n$ is the set of all states
Rectangular automaton

Definition

A rectangular automaton $\mathcal{A}$ is a tuple $\mathcal{H} = (\text{Loc}, \text{Var}, \text{Con}, \text{Lab}, \text{Edge}, \text{Act}, \text{Inv}, \text{Init})$ with

- finite set of locations $\text{Loc}$,
- finite set of real-valued variables $\text{Var} = \{x_1, \ldots, x_n\}$,
- a function $\text{Con} : \text{Loc} \to 2^{\text{Var}}$ assigning controlled variables to locations,
- finite set of synchronization labels $\text{Lab}$,
- finite set of edges $\text{Edge} \subseteq \text{Loc} \times \text{Lab} \times \mathbb{R}^n \times \mathbb{R}^n \times 2^{\{1, \ldots, n\}} \times \text{Loc}$,
- a flow function $\text{Act} : \text{Loc} \to \mathbb{R}^n$,
- an invariant function $\text{Inv} : \text{Loc} \to \mathbb{R}^n$,
- initial states $\text{Init} : \text{Loc} \to \mathbb{R}^n$.

- **States:** $\sigma = (l, \vec{x}) \in (\text{Loc} \times \mathbb{R}^n)$ with $\vec{x} \in \text{Inv}(l)$
- **State space:** $\Sigma \subseteq \text{Loc} \times \mathbb{R}^n$ is the set of all states
- Is the state space rectangular?
Rectangular automaton

- **Flows**: first time derivatives of the flow trajectories in location \( l \in \text{Loc} \) are within \( \text{Act}(l) \)

- **Jumps**: \( e = (l, a, \text{pre}, \text{post}, \text{jump}, l') \in \text{Edge} \) may move control from location \( l \) to location \( l' \) starting from a valuation in \( \text{pre} \), changing the value of each variable \( x_i \) to a nondeterministically chosen value from \( \text{post}_i \) (the projection of \( \text{post} \) to the \( i \)th dimension), such that the values of the variables \( x_i \notin \text{jump} \) are unchanged.
Operational semantics
Operational semantics

\[(l, a, pre, post, jump, l') \in Edge\]
\[\vec{x} \in pre \quad \vec{x}' \in post \quad \forall i \notin jump. \quad x'_i = x_i \quad \vec{x}' \in Inv(l')\]

\[(l, \vec{x}) \xrightarrow{a} (l', \vec{x}')\]

Rule Discrete
Operational semantics

\[(l, a, \text{pre}, \text{post}, \text{jump}, l') \in \text{Edge} \]
\[
\vec{x} \in \text{pre} \quad \vec{x}' \in \text{post} \quad \forall i \notin \text{jump}. \quad x'_i = x_i \quad \vec{x}' \in \text{Inv}(l')
\]

\[
(l, \vec{x}) \xrightarrow{a} (l', \vec{x}')
\]

**Rule Discrete**

\[
(t = 0 \land \vec{x} = \vec{x}') \lor (t > 0 \land (\vec{x}' - \vec{x})/t \in \text{Act}(l)) \quad \vec{x}' \in \text{Inv}(l)
\]

\[
(l, \vec{x}) \xrightarrow{t} (l, \vec{x}')
\]

**Rule Time**
Operational semantics

\[(l, a, pre, post, jump, l') \in Edge\]
\[\vec{x} \in pre \quad \vec{x}' \in post \quad \forall i \notin jump. \ x'_i = x_i \quad \vec{x}' \in Inv(l')\]

\[\begin{array}{l}
(l, \vec{x}) \xrightarrow{a} (l', \vec{x}')
\end{array}\]

\[\begin{array}{l}
(t = 0 \land \vec{x} = \vec{x}') \lor (t > 0 \land (\vec{x}' - \vec{x})/t \in Act(l)) \quad \vec{x}' \in Inv(l)
\end{array}\]

\[\begin{array}{l}
(l, \vec{x}) \xrightarrow{t} (l, \vec{x}')
\end{array}\]

- Execution step: \(\rightarrow = a \cup t\)
- Path: \(\sigma_0 \rightarrow \sigma_1 \rightarrow \sigma_2 \ldots \) with \(\sigma_0 = (l_0, \vec{x}_0), \ \vec{x}_0 \in Inv(l_0)\)
- Initial path: path \(\sigma_0 \rightarrow \sigma_1 \rightarrow \sigma_2 \ldots \) with \(\sigma_0 = (l_0, \vec{x}_0), \ \vec{x}_0 \in Init(l_0)\)
- Reachability of a state: exists an initial path leading to the state
Example rectangular automaton

\[ x = 0 \]

- \( l_1 \)
  - \( \dot{x} \in [1, 2] \)
  - \( x \leq 6 \)

- \( l_2 \)
  - \( \dot{x} \in [-4, -2] \)

- \( l_4 \)
  - \( \dot{x} \in [1, 2] \)
  - \( x \leq 4 \)

\[ d \]

- \( x \geq 0 \)
  - \( a \) \( x \geq 2 \rightarrow x := 4 \)
  - \( x = 0 \rightarrow x := [-2, -1] \)

\[ c \]

- \( b \) \( x \leq -2 \rightarrow x := [0, 4] \)
If we replace rectangular sets with linear sets, we obtain linear hybrid automata, a super-class of rectangular automata.

A timed automaton is a special rectangular automaton.
If we replace rectangular sets with linear sets, we obtain linear hybrid automata, a super-class of rectangular automata.

A timed automaton is a special rectangular automaton.

This class lies at the boundary of decidability.
The reachability problem is decidable for initialized rectangular automata:
The reachability problem is decidable for initialized rectangular automata:

**Definition**

A rectangular automaton $A$ is initialized, if for every edge $(l, a, pre, post, jump, l')$ of $A$, and every variable index $i \in \{1, \ldots, n\}$ with $Act(l)_i \neq Act(l')_i$, we have that $i \in jump$.

The reachability problem becomes undecidable if one of the restrictions is relaxed.
This rectangular automaton is initialized.
What we already know

A **timed automaton** is a special rectangular automaton such that

- for each edge, \( post_i \) is a single value for each \( i \in \text{jump} \) and
- every variable is a **clock**, i.e., \( Act(l)(x) = [1, 1] \) for all locations \( l \) and variables \( x \).
What we already know

A timed automaton is a special rectangular automaton such that
- for each edge, $post_i$ is a single value for each $i \in jump$ and
- every variable is a clock, i.e., $Act(l)(x) = [1, 1]$ for all locations $l$ and variables $x$.

**Lemma**

*The reachability problem for timed automata is complete for PSPACE.*
Lemma

The reachability problem for initialized rectangular automata is complete for PSPACE.
Decidability results

**Lemma**

The reachability problem for initialized rectangular automata is complete for PSPACE.

Timed automaton

\[\uparrow\]

Initialized stopwatch automaton

\[\uparrow\]

Initialized singular automaton

\[\uparrow\]

Initialized rectangular automaton
Decidability results

Timed automaton

Initialized stopwatch automaton
A stopwatch is a variable with derivatives 0 or 1 only.

A stopwatch automaton is as a timed automaton but allowing stopwatch variables instead of clocks.

Initialized stopwatch automata can be polynomially encoded by timed automata.

**Lemma**

*The reachability problem for initialized stopwatch automata is complete for PSPACE.*

However, the reachability problem for non-initialized stopwatch automata is undecidable.
Proof idea:
Proof idea: Notice, that a timed automaton is a stopwatch automaton such that every variable is a clock.

Assume that $C$ is an $n$-dimensional initialized stopwatch automaton. Let $\kappa_C$ be the set of constants used in the definition of $C$, and let $\kappa_- = \kappa_C \cup \{-\}$.

We define an $n$-dimensional timed automaton $D_C$ with locations $\text{Loc}_{D_C} = \text{Loc}_C \times \kappa^{1,\ldots,n}$. Each location $(l, f)$ of $D_C$ consists of a location $l$ of $C$ and a function $f : \{1, \ldots, n\} \rightarrow \kappa_-$. Each state $q = ((l, f), \vec{x})$ of $D_C$ represents the state $\alpha(q) = (l, \vec{y})$ of $C$, where $y_i = x_i$ if $f(i) = -$, and $y_i = f(i)$ if $f(i) \neq -$.

Intuitively, if the $i$th stopwatch of $C$ is running (slope 1), then its value is tracked by the value of the $i$th clock of $D_C$; if the $i$th stopwatch is halted (slope 0) at value $k \in \kappa_C$, then this value is remembered by the current location of $D_C$. 
Decidability results

Timed automaton
↑
Initialized stopwatch automaton
↑
Initialized singular automaton
A variable $x_i$ is a **finite-slope variable** if $\text{flow}(l)_i$ is a singleton in all locations $l$.

A **singular automaton** is as a stopwatch automaton but allowing finite-slope variables instead of stopwatches.

Initialized singular automata can be polynomially encoded by initialized stopwatch automata.

**Lemma**

*The reachability problem for initialized singular automata is complete for PSPACE.*
Proof idea:
Proof idea: Let $B$ be an $n$-dimensional initialized singular automaton. We define an $n$-dimensional initialized stopwatch automaton $C_B$ with the same location set, edge set, and label set as $B$.

Each state $q = (l, \vec{x})$ of $C_B$ corresponds to the state $\beta(q) = (l, \beta(\vec{x}))$ of $B$ with $\beta : \mathbb{R}^n \rightarrow \mathbb{R}^n$ defined as follows:

For each location $l$ of $B$, if $\text{Act}_B(l) = \Pi_{i=1}^n [k_i, k_i]$, then

$$\beta(x_1, \ldots, x_n) = (l_1 \cdot x_1, \ldots, l_n \cdot x_n)$$

with $l_i = k_i$ if $k_i \neq 0$, and $l_i = 1$ if $k_i = 0$;

$\beta$ can be viewed as a rescaling of the state space. All conditions in the automaton $B$ occur accordingly rescaled in $C_B$.

We have:

- The reachable set of $\text{Reach}(B)$ of $B$ is $\beta(\text{Reach}(C_B))$. 
Decidability results

Timed automaton
↑
Initialized stopwatch automaton
↑
Initialized singular automaton
↑
Initialized rectangular automaton
Lemma

The reachability problem for initialized rectangular automata is complete for PSPACE.
Proof idea:
Proof idea: An $n$-dimensional initialized rectangular automaton $A$ can be translated into a $2n$-dimensional initialized singular automaton $B$, such that $B$ contains all reachability information about $A$.

The translation is similar to the subset construction for determinizing finite automata.

The idea is to replace each variable $c$ of $A$ by two finite-slope variables $c_l$ and $c_u$: the variable $c_l$ tracks the least possible value of $c$, and $c_u$ tracks the greatest possible value of $c$. 